New Theoretical and Computational Results For Regular Languages*

Chia-Hsiang Chang and Robert Paige
email: changch@cs.nyu.edu, paige@cs.nyu.edu
New York University/Courant Institute
251 Mercer St. New York, NY 10012

October 14 1991

Abstract

We show how to turn a regular expression into an $O(s)$ space representation of McNaughton and Yamada's NFA, where s is the number of NFA states. The standard adjacency list representation of McNaughton and Yamada's NFA takes up $s+s^2$ space in the worst case. The adjacency list representation of the NFA produced by Thompson takes up between $2r$ and $5r$ space, where $r \geq s$ in general, and can be arbitrarily larger than s. Given any set T of NFA states, our representation can be used to compute the set N of states one transition away from the states in T in optimal time $O(|T| + |N|)$. McNaughton and Yamada's NFA requires $\Theta(|T| \times |N|)$ in the worst case. Using Thompson's NFA, the equivalent calculation requires $\Theta(r)$ time in the worst case.

An implementation of our NFA representation confirms that it takes up an order of magnitude less space than McNaughton and Yamada's machine. An implementation to produce a DFA from our NFA representation by subset construction shows linear and quadratic speedups over subset construction starting from both Thompson's and McNaughton and Yamada's NFA's. It also shows that the DFA produced from our NFA is as much as one order of magnitude smaller than DFA's constructed from the two other NFA's.

1 Introduction

The growing importance of regular languages and their associated computational problems in languages and compilers is underscored by the granting of the Turing Award to Rabin and Scott in 1976, in part, for their ground breaking logical and algorithmic work in regular languages [16]. Of special significance was

*This research was partially supported by Office of Naval Rsearch Grant No. N00014-90-J-1890 and Air Force Office of Scientific Research Grant No. AFOSR-91-0308.

their construction of the canonical minimum state DFA that had been described nonconstructively in the proof of the Myhill-Nerode Theorem[14,15]. Rabin and Scott's work, which was motivated by theoretical considerations, has gained in importance as the number of practical applications has grown. In particular, the construction of finite automata from regular expressions is of central importance to the compilation of communicating processes[4], string pattern matching[3], model checking[8], lexical scanning[2], and VLSI layout design[20]; unit time incremental acceptance testing in a DFA is also a crucial step in LR_k parsing[12]; algorithms for acceptance testing and DFA construction from regular expressions are implemented in the UNIX operating system[17].

Throughout this paper our model of computation is a uniform cost sequential RAM [1]. We report the following four results.

1. Recently Berry and Sethi[5] used results of Brzozowski[6] to formally derive and improve McNaughton and Yamada's algorithm[13] for turning regular expressions into NFA's. NFA's produced by this algorithm have fewer states than NFA's produced by Thompson's algorithm[18], and in practice they are known to outperform Thompson's NFA's for acceptance testing. Berry and Sethi's algorithm has two passes and can easily be implemented to run in time $\Theta(m)$ and auxiliary space $\Theta(r)$, where r is the length of the regular expression, and m is the number of edges in the NFA produced. We present an algorithm that computes the same NFA in a single left-to-right scan over the regular expression. It runs in the same asymptotic time $\Theta(m)$ as Berry and Sethi, but it improves the auxiliary space to $\Theta(s)$, where s is the number of occurrences of alphabet symbols appearing in the regular expression.

2. One disadvantage of McNaughton and Yamada's NFA is that its worst case number of edges is $m = \Theta(s^2)$, which is also a worst case space bound for the standard adjacency list implementation. Thompson's NFA only has between r and $2r$ states and between r and $3r$ edges. We introduce a new compressed representation for McNaughton and Yamada's NFA that uses only $\Theta(s)$ space. Our compressed NFA can be constructed from a regular expression R in $\Theta(r)$ time and $O(s)$ auxiliary space. It supports acceptance testing in worst-case time $O(s|x|)$ for arbitrary string x, and a promising new way to construct DFA's faster than the classical subset construction of Rabin and Scott.

3. Our main theoretical result is a proof that the compressed NFA can be used to compute the set of states N one edge away from an arbitrary set of states T in McNaughton and Yamada's NFA in optimal time $O(|T| + |N|)$. The previous best worst-case time is $\Theta(|T| \times |N|)$.

4. We give empirical evidence that our algorithm for NFA acceptance testing using the compressed NFA yields a constant factor speedup over acceptance testing using Thompson's NFA, and is comparable to McNaughton and Yamada's NFA. We give more dramatic empirical evidence that constructing a DFA from our compressed NFA can be achieved in time one order of magnitude faster than the classical Rabin and Scott subset construction (cf. Chapter 3 of [2]) starting from either Thompson's NFA or

McNaughton and Yamada's NFA. Our benchmarks also show subset construction being faster when it starts from Thompson's machine than from McNaughton and Yamada's NFA.

The next two sections present standard terminology and background material, and can be skipped by anyone who knows Chapter 3 of [2]. Section 4 reformulates McNaughton and Yamada's algorithm from an automata theoretic point of view. Section 5 describes our new algorithm to turn a regular expression into McNaughton and Yamada's NFA. In Section 6 we show how to construct a compressed form of this NFA. Analysis of the compressed NFA is presented in Theorem 6, which is our main theoretical result. In section 7, we show how to further compressed our NFA. Section 8 discusses experimental results showing how our compressed NFA compares with other NFA's in solving acceptance testing and DFA construction. Section 9 mentions future research.

2 Terminology

The following basic definitions and terminology can be found in [10]. By an *alphabet* we mean a finite nonempty set of symbols. If Σ is an alphabet, then Σ^* denotes the set of all finite strings of symbols in Σ. If Σ is an alphabet, then any subset of Σ^* is a *language* over Σ. If L_1 and L_2 are two languages, then the cross product $L_1\,L_2 = \{xy : x \in L_1, y \in L_2\}$ represents the set of all strings $x\,y$ that result from concatenating each $x \in L_1$ with each $y \in L_2$. If λ stands for the empty string, and $\emptyset$ represents the empty set, then $L\,\{\lambda\} = \{\lambda\}\,L = L$, and $L\,\emptyset = \emptyset\,L = \emptyset$ for any language L.

Definition 1 *Let L_R be the language denoted by regular expression R. Let Σ be a finite alphabet. Then the regular expressions are the smallest set of terms that contains*

- *$\emptyset$ (which represents the empty set)*
- *λ (which represents the set $\{\lambda\}$), where λ is the empty string*
- *a (which represents the set $\{a\}$) for each symbol $a \in \Sigma$*
- *$T|S$ (which represents the set $L_T \cup L_S$), where T and S are regular expressions*
- *TS (which represents the cross product set $L_T\,L_S$), where R and S are regular expressions*
- *T^* (which represents lfp $S.\{\lambda\} \cup L_T S$, where lfp $X.E(X)$ is the minimum value X such that $X = E(X)$), where T is a regular expression.*

A *nondeterministic finite automata* (abbr. NFA) M is a 5-tuple $(\Sigma, Q, I, F, \delta)$, where Q is a set of states, $I \subseteq Q$ is the set of initial states, $F \subseteq Q$ is the set of final states, and $\delta \subseteq Q \times (\Sigma \times Q)$ is a labeled directed graph with vertices Q and an edge labeled a connecting state q to state p for every $[q,[a,p]]$ belonging to δ. For all $q \in Q$ and $a \in \Sigma$ we use the notation $\delta(q,a)$ to denote the set $\{p : [q,[a,p]] \in \delta\}$ of all states

reachable from state q by a single edge labeled 'a'. It is useful to generalize this notation with the following rules, where $T \subseteq Q$, $s \in \Sigma^*$, and $B \subseteq \Sigma^*$:

$$\begin{aligned} \delta(T, a) &= \cup_{q \in T} \delta(q, a) \\ \delta(q, as) &= \delta(\delta(q, a), s) \\ \delta(T, s) &= \cup_{q \in T} \delta(q, s) \\ \delta(T, B) &= \cup_{b \in B} \delta(T, b) \end{aligned}$$

The language L accepted by M, denoted by $L(M)$, is defined by the rule

$$s \in L \leftrightarrow \delta(I, s) \cap F \neq \emptyset \tag{1}$$

In other words, $L = \{s \in \Sigma^* | \delta(I, s) \cap F \neq \emptyset\}$. M is a *deterministic finite automata* (abbr. DFA) if graph δ has no more than one edge with the same label leading out from each vertex, and if I contains exactly one state. Regular expressions and NFA's that represent the same regular language are said to be *equivalent*.

3 Background

Kleene [11] characterized regular languages equivalently in terms of languages denoted by regular expressions and languages accepted by DFA's. Rabin and Scott [16] showed that NFA's also characterize the regular languages, and their work led to algorithms to decide whether an arbitrary string is accepted by an NFA.

Let n be the number of NFA states, m be the number of edges, and k be the alphabet size. For an NFA represented by an adjacency matrix of size n^2 for each alphabet symbol, acceptance testing takes $O(n|x|)$ bit vector operations and $O(n)$ auxiliary space. Alternatively, for an NFA implemented by an adjacency list of size m with a perfect hash table [9] storing the alphabet symbols at each state, this test takes time proportional to $m|x|$ in the worst case. For DFA's the same data structure leads to a better time bound of $\theta(|x|)$. However, there are NFA's for which the smallest equivalent DFA (unique up to isomorphism of state labels as shown by Myhill [14] and Nerode [15]) has an exponentially greater number of states. Thus, the choice between using an NFA or DFA is a space/time tradeoff.

There are two main approaches for turning regular expressions into equivalent NFA's. One, due to Thompson [18], constructs an NFA (augmented with λ edges) in which the number n of states is somewhere between the length r of the regular expression and $2r$, and the outdegree of any state is no greater than 2. Consequently $m = O(n)$, and the adjacency list implementation does not even require perfect hashing to preserve the $O(n|x|)$ time bound. Thompson's construction is a simple, bottom-up, method that processes the regular expression as it is parsed. The time and space is linear in r.

Another approach, based on Berry and Sethi's [5] improvement to McNaughton and Yamada [13], constructs an NFA in which the number n of states is precisely one plus the number s of occurrences of alphabet symbols appearing in the regular expression. In general, s can be arbitrarily smaller than r.

For the bit matrix representation, McNaughton and Yamada's NFA can be used to solve acceptance testing using $O(s|x|)$ bit vector operations, which is superior to the time bound for Thompson's machine. With the adjacency list representation the worst case number of edges $m = \Omega(s^2)$ leads to a worst case time bound $\Theta(m|x|)$ which is one order of magnitude worse than the time bound for Thompson's machine. However, the fact that McNaughton and Yamada's NFA is a DFA when all of the alphabet symbols are distinct may explain, in part, why it is observed to outperform Thompson's NFA for a large subclass of the instances. The Berry/Sethi construction scans the regular expression twice, and, with only a little effort, both passes can be made to run in linear time and auxiliary space with respect to r plus the size of the NFA (for either adjacency list or matrix implementations).

There is one main approach for turning NFA's (constructed by either of the two methods above) into DFA's. This is by the Rabin and Scott subset construction [16].

4 McNaughton and Yamada's NFA

It is convenient to reformulate McNaughton and Yamada's transformation from regular expressions to NFA's[13] in the following way.

Definition 2 *A normal NFA (abbr. NNFA) is an NFA with one starting state q_0 having no edges leading into it, and all edges leading into each state are labeled with the same symbol. For an NNFA with alphabet Σ the transition map is represented by a binary edge relation $\delta \subseteq Q \times Q$ and assignment $A : (Q - \{q_0\}) \rightarrow \Sigma$, where $A(q)$ is the label assigned to every edge leading into state q.*

Definition 3 *If $M = (\Sigma, Q, q_0, F, \delta, A)$ is an NNFA, then $tail(M) = (\Sigma, Q - \{q_0\}, \delta\{q_0\}, F - \{q_0\}, \{[q,t] \in \delta | q \neq q_0\}, A)$.*

It is a desirable and obvious fact (which follows immediately from the definition of an NNFA) that when A is one-to-one, then no state can have more than one transition with the same label. Hence, such an NNFA is a DFA.

We can implement McNaughton and Yamada's algorithm to turn a regular expression R into an NNFA while performing a single left-to-right shift/reduce parse of R (but without actually producing a parse tree). To explain how this is done, we use the notational convention that M_R denotes an NFA equivalent to regular expression R. Each time a subexpression S of R is reduced during parsing, $tail(M_S)$ is computed, where M_S is an NNFA equivalent to S. The last step computes an NNFA M_R from $tail(M_R)$. However, M_R cannot be computed from $tail(M_R)$ unless we know whether M_R accepts λ, which indicates whether or not the start state for M_R is a final state.

Regular expressions are *restricted* if $\emptyset$ is not a subexpression. There is a linear time algorithm to convert regular expressions into their equivalent restricted forms. Without loss of generality, we will assume

throughout this paper that regular expressions are restricted.

Let $null_R = \{\lambda\}$ if $\lambda \in L_R$; otherwise, let $null_R = \emptyset$. If $tail(M_R) = (\Sigma, Q, I, F, \delta, A)$, and $q_0 \notin Q$, then the following formula

$$M_R = (\Sigma, Q \cup \{q_0\}, \{q_0\}, F \cup (\{q_0\} null_R), \delta \cup \{[q_0, y] : y \in I\}, A) \tag{2}$$

indicates how to compute M_R from $tail(M_R)$ and $null_R$.

Theorem 1 *(McNaughton and Yamada) Given any regular expression R with s occurrences of alphabet symbols from Σ, we can construct an NNFA M_R with $s+1$ states.*

Proof The proof uses structural induction to show that for any regular expression R, we can always compute $tail(M_R)$ and $null_R$ for some NNFA M_R. Then equation (2) can be used to obtain M_R. We assume a fixed alphabet Σ. There are two base cases, which are easily verified.

$$tail(M_\lambda) = (Q_\lambda = \emptyset, \delta_\lambda = \emptyset, A_\lambda = \emptyset, I_\lambda = \emptyset, F_\lambda = \emptyset, null_\lambda = \{\lambda\}) \tag{3}$$

$$tail(M_a) = (Q_a = \{q_0\}, I_a = \{q_0\}, F_a = \{q_0\}, \delta_a = \emptyset, A_a = \{[q_0, a]\}, null_a = \emptyset), \tag{4}$$
$$\text{where } a \in \Sigma, \text{and } q_0 \text{ is a new state}$$

To use induction, we assume that T and S are two arbitrary regular expressions equivalent respectively to NNFA's M_T and M_S with $tail(M_T) = (Q_T, I_T, F_T, \delta_T, A_T)$ and $tail(M_S) = (Q_S, I_S, F_S, \delta_S, A_S)$, where Q_T and Q_S are disjoint. Then we can easily verify that

$$tail(M_{T|S}) = (Q_{T|S} = Q_T \cup Q_S, \delta_{T|S} = \delta_T \cup \delta_S, A_{T|S} = A_T \cup A_S, I_{T|S} = I_T \cup I_S,$$
$$F_{T|S} = F_T \cup F_S, null_{T|S} = null_T \cup null_S) \tag{5}$$

$$tail(M_{TS}) = (Q_{TS} = Q_T \cup Q_S, \delta_{TS} = \delta_T \cup \delta_S \cup F_T I_S, A_{TS} = A_T \cup A_S,$$
$$I_{TS} = I_T \cup null_T I_S, F_{TS} = F_S \cup null_S F_T, null_{TS} = null_T null_S) \tag{6}$$

$$tail(M_{T^*}) = (Q_{T^*} = Q_T, \delta_{T^*} = \delta_T \cup F_T I_T, A_{T^*} = A_T, I_{T^*} = I_T, F_{T^*} = F_T,$$
$$null_{T^*} = \{\lambda\}) \tag{7}$$

Disjointness of the unions used to form the set of states for the cases $T|S$ and TS proves the assertion about the number of states. We can convert $tail(M_R)$ into M_R using formula (2) □

The proof of Theorem 1 leads to McNaughton and Yamada's algorithm. The construction of label map A shows that when all of the occurrences of alphabet symbols appearing in the regular expression contain distinct symbols, then A is one-to-one. In this case, a DFA would be produced.

Analysis determines that this algorithm falls short of optimal performance, because the operation $\delta_T \cup F_T I_T$ within formula (7) for $tail(M_{T^*})$ is not disjoint; all other unions are disjoint and can be implemented

in unit time. In particular, this overlapping union makes McNaughton and Yamada's algorithm use time $\theta(m\sqrt{m}logm)$ to transform regular expression

$$(((((a_1^*|a_2)^*|a_3)^*...a_k)^* \tag{8}$$

into an NNFA with $k+1$ states and $m = k^2$ edges.

5 Faster NFA Construction

By recognizing the overlapping union $\delta_T \cup F_T I_T$ within formula (7) for $tail(M_{T*})$ as the source of inefficiency, we can maintain invariant $nred_T = F_T I_T - \delta_T$ in order to replace the overlapping union by the equivalent disjoint union $\delta_T \cup nred_T$. In order to maintain $nred_T$ as a component of the $tail$ NNFA computation given above, we can use the following recursive definition, obtained by simplifying expression $F_R I_R - \delta_R$ and using the rules from the proof of Theorem 1.

$$nred_\lambda = \emptyset \tag{9}$$

$$nred_a = F_a I_a, where\ a \in \Sigma \tag{10}$$

$$nred_{T|S} = nred_T \cup nred_S \cup F_T I_S \cup F_S I_T \tag{11}$$

$$nred_{TS} = F_S I_T \cup null_S nred_T \cup null_T nred_S \tag{12}$$

$$nred_{S*} = \emptyset \tag{13}$$

Rules (9), (10) and (13) are trivial. Rule (11) follows from applying distributive laws to simplify formula

$$nred_{T|S} = (F_T \cup F_S)(I_T \cup I_S) - (\delta_T \cup \delta_S)$$

Rule (12) is obtained by applying distributed laws to simplify formula,

$$nred_{TS} = (F_S \cup null_S F_T)(I_T \cup null_T I_S) - (\delta_T \cup \delta_S \cup F_T I_S)$$

Each union operation is disjoint and, hence, $O(1)$ time implementable. However, there is a serious loss of efficiency computing cartesian products in rules (11) and (12). Such products do not contribute edges to the NNFA for regular expressions TS when these products belong to $nred_T$ and $null_S$ is empty, or when they belong to $nred_S$ and $null_T$ is empty.

To overcome this problem we will use lazy evaluation to compute cartesian products only when they actually contribute edges to the NNFA. Thus, instead of maintaining a union $nred_R$ of cartesian products, we will maintain a set $lazynred_R$ of pairs of sets. Consequently, the overlapping union $\delta_T \cup F_T I_T$ within formula (7) for $tail(M_{T*})$ can be replaced by

$$\delta_T \cup (\cup_{[A,B] \in lazynred_T} A \times B) \tag{14}$$

However, this solution creates another problem: the sets forming F and I, which are computed by the rules to construct the *tail* of an NNFA, must be persistent in the following sense. Let the sets in the sequence forming F (respectively I) be called *F-sets* (respectively *I-sets*). Each *F-set* (respectively *I-set*) could be stored as a first (respectively second) component of a pair belonging to *lazynred*. Given any such pair, we need to iterate through the *I-set* S stored in the second component of the pair in $O(|S|)$ time.

The sequence of *F-sets* (respectively *I-sets*) are formed by two operations: 1. create a new singleton set; and 2. form a new set by taking the disjoint union of two previous sets in the sequence. Clearly, each of these sequences can be stored as a binary forest in which each subtree in the forest represents a set in the sequence, where the elements of the set are stored in the frontier. By construction each internal node in the forest has two children.

We call the forest storing the *F-sets* (respectively *I-sets*) the *F-forest* (respectively *I-forest*). For each node n belonging to the *F-forest* (respectively *I-forest*), let $Fset(n)$ (respectively $Iset(n)$) denote the *F-set* (respectively *I-set*) represented by n.

Each node in the *F-forest* and *I-forest* except the roots stores a parent pointer. Each node n in the *I-forest* also stores a pointer to the leftmost leaf of the subtree rooted in n and a pointer to the rightmost leaf of the subtree rooted n. The frontier nodes of the *I-forest* are linked.

This data structure preserves the unit-time disjoint union for *F-sets* and *I-sets*, and supports linear time iteration through the frontier of any node in the *I-forest*. Since all the *F-sets* and *I-sets* are subsets of the NFA states Q, the *F-forest* and *I-forest* each is stored in $O(|Q|)$ space.

Theorem 2 *For any regular expression R we can compute $lazynred_R$ in time $O(r)$ and auxiliary space $O(s)$, where r is the size of regular expression R, and s is the number of occurrences of alphabet symbols appearing in R.*

Proof If T and S are two sets, let $pair(T,S) = \{[T,S]\}$ if both T and S are nonempty; otherwise, let $pair(T,S) = \emptyset$. The proof makes use of the following recursive definition of $lazynred_R$ obtained from the recursive definition of $nred_R$.

$$lazynred_\lambda = \emptyset \tag{15}$$

$$lazynred_a = pair(F_a, I_a), where\ a \in \Sigma \tag{16}$$

$$lazynred_{T|S} = lazynred_T \cup lazynred_S \cup pair(F_T, I_S) \cup pair(F_S, I_T) \tag{17}$$

$$lazynred_{TS} = pair(F_S, I_T) \cup null_S lazynred_T \cup null_T lazynred_S \tag{18}$$

$$lazynred_{S*} = \emptyset \tag{19}$$

Operation $pair(T,S)$ takes unit time and space. Each union operation occurring in the rules above is disjoint and, hence, implementable in unit time. Rule (16) contributes unit time and space for each alphabet symbol occurring in R, or $O(s)$ time and space overall. Rule (17) contributes unit time for each alternation operator

appearing in R or $O(r)$ time overall. It contributes two units space for each alternation operator both of whose alternands contain at least one alphabet symbol. Hence, the overall space contributed by this rule is less than $2s$. By a similar argument, Rule (18) contributes $O(r)$ time and less than s space overall. The other two rules contribute no more than $O(r)$ time overall. Hence, the time and space needed to compute $lazynred_R$ is $O(r)$ and $O(s)$ respectively $\square$

By Theorems 1 and 2, and by the fact that $nred_R$ can be computed from $lazynred_R$ in $O(|nred_R|)$ time using formula (14), we have our first theoretical result.

Theorem 3 *For any regular expression R we can compute an equivalent NNFA with $s+1$ states in time $O(r+m)$ and auxiliary space $O(s)$, where r is the size of regular expression R, m is the number of edges in the NNFA, and s is the number of occurrences of alphabet symbols appearing in R.*

6 Improving Space for McNaughton and Yamada's NFA

Theorem 3 leads to a new algorithm that computes the adjacency form of the NNFA in a single left-to-right shift/reduce parse of the regular expression R. Although this improves upon the algorithm of Berry and Sethi, McNaughton and Yamada's NNFA has certain theoretical disadvantages over simpler Thompson's NFA. Recall from example (8) that the number of edges in McNaughton and Yamada's machine can be the square of the number of edges in Thompson's machine (since Thompson's NFA has $m = O(n)$). Consequently, Thompson's NFA is likely to be more desirable in time and space for DFA construction by subset construction when the adjacency list implementation is used. We also believe that the bit vector implementation will rarely be more desirable than the compact adjacency list implementation.

Nevertheless, we can modify the algorithm just given so that in $O(r)$ time it produces an $O(s)$ space compressed NFA that encodes McNaughton and Yamada's NNFA, and that supports acceptance testing in $O(s|x|)$ time. In the same way that $nred_R$ was represented more compactly as $lazynred_R$, we can represent δ_R, which is a union of cartesian products, as a set $lazy\delta_R$ of pairs of set-valued arguments of these products. If M_R is the NNFA equivalent to regular expression R, then the rules for $tail(M_R)$ are given just below:

$$lazy\delta_\lambda = \emptyset \tag{20}$$

$$lazy\delta_a = \emptyset \tag{21}$$

$$lazy\delta_{T|S} = lazy\delta_T \cup lazy\delta_S \tag{22}$$

$$lazy\delta_{TS} = pair(F_T, I_S) \cup lazy\delta_T \cup lazy\delta_S \tag{23}$$

$$lazy\delta_{S^*} = lazy\delta_S \cup lazynred_S \tag{24}$$

After the preceding rules are processed we can obtain a representation for M_R by introducing a new state q_0 and by adding tuple $[q_0, I_R]$ to $lazy\delta$ in accordance with formula (2).

Consequently, if T is a subset of the NFA states Q, then we can compute the collection of sets $\delta(T, a)$ for all of the alphabet symbols $a \in \Sigma$ as follows. First we compute

$$finddomain(T) = \{X : [X, Y] \in lazy\delta | T \cap X \neq \emptyset\}$$

which is used to find the set of next states

$$next\_states(T) = \{Y : [X, Y] \in lazy\delta | X \in finddomain(T)\}$$

Finally, for each alphabet symbol $a \in \Sigma$, we see that

$$\delta(T, a) = \{q : Y \in next\_states(T), q \in Y | A(q) = a\}$$

In order to explain how $lazy\delta$ is implemented, we will use some additional terminology. For each *F-set* S represented by node n in the *F-forest*, n stores a pointer to a list of nodes in the *I-forest* representing set $\{Y : [S, Y] \in lazy\delta\}$. Furthermore, the *F-forest* and *I-forest* are compressed to only store nodes representing sets that appear as the first or second components of a pair $[X, Y] \in lazy\delta$. In other words, we make $lazy\delta$ a total onto binary relation. This can be achieved on-line as the *F-forest* and *I-forest* are constructed by a kind of path compression that affects the preprocessing time and space by no more than a small constant factor. Thus, we have

Theorem 4 *For any regular expression R, its equivalent compressed NFA, consisting of F-forest, I-forest and $lazy\delta$, takes up $O(s)$ space and can be computed in time $O(r)$ and auxiliary space $O(s)$.*

Proof Since each internal node in the *F-forest* and *I-forest* have at least two children, and since their leaves are distinct occurrences of alphabet symbols, they take up $O(s)$ space. Each of the unions in the rules to compute $lazy\delta$ is disjoint, and hence takes unit time. By the same argument used to analyze the overall space contributed by Rule (18) in the proof of Theorem 2, we see that Rule (23) contributes $O(s)$ space and $O(r)$ overall to $lazy\delta$. By Rule (19), Theorem 2, and a simple application of structural induction, we also see that the space contributed by Rule (24) (which results from adding $lazynred$ to $lazy\delta$) overall is $O(s)$. The overall time bound for each rule is easily seen to be $O(r)$ □

The compressed NFA also supports an efficient evaluation of the three preceding queries in order to simulate transition map δ. The best previous worst case time bound for inputing a subset T of states and computing the collection of sets $\delta(T, a)$ for all of the alphabet symbols $a \in \Sigma$ is $\Theta(|T| \times |\delta(T, \Sigma)|)$ using an adjacency list implementation of McNaughton and Yamada's NFA, or $\Theta(r)$ using Thompson's NFA.

In Theorem 6 we improve this bound, and obtain, essentially, optimal asymptotic time without exceeding $O(s)$ space. This is our main theoretical result. It explains the apparent superior performance of acceptance testing using our compressed NFA over Thompson's. It explains more convincingly why constructing a DFA starting from our compressed NFA is at least one order of magnitude faster than when we start from either Thompson's or McNaughton and Yamada's NFA. These empirical results are presented in section 8.

Before proving the theorem, we will first prove the following technical lemma.

Lemma 5 *Let T be a set of states in the compressed NNFA built from regular expression R, and $lazy\delta_T = \{[X,Y] : [X,Y] \in lazy\delta | X \cap T \neq \emptyset\}$. Then $|lazy\delta_T| = O(|T| + |\delta(T,\Sigma)|)$.*

Proof The result follows from proving that $O(|T| + |\delta(T,\Sigma)|)$ is a bound for each of the subsets of $lazy\delta_T$ contributed by rules (16), (17), (18), and (23) respectively. The bound holds for subsets contributed by rules (16), (17), and (23), because they form one-to-one maps.

The proof for the subset contributed by (18) is split into two cases. For convenience, let T_Q denote the set of states in T such that their corresponding symbol occurrences appear in regular expression Q, where Q is a subexpression of R. First, consider the set A of pairs $[F_S, I_Q] \in lazy\delta_T$ for subexpressions QS, where $T_Q = \emptyset$. We claim that these edges form a one-to-many map, which implies the bound. Suppose this were not the case. Then we would have a subexpression QS, and a subexpression LP of Q such that $I_Q = I_L$ and pairs $[F_S, I_Q]$ and $[F_P, I_L]$ belonging to A. However, since Q contains no occurrence of an alphabet symbol in T, then P does not either. Hence, the pair $[F_P, I_L]$ cannot belong to A. Hence, the claim holds.

Next, consider the set B of pairs $[F_S, I_Q] \in lazy\delta_T$ for subexpressions QS, where $T_Q \neq \emptyset$. Proceeding from inner-most to outer-most subexpression QS, we charge each pair $[F_S, I_Q] \in B$ to an uncharged state in T_Q. A simple structural induction would show that T_Q contains at least one uncharged state. Let LP be an inner-most subexpression in R such that T_L is nonempty, and $[F_P, I_L] \in lazy\delta_T$. Then both T_L and T_P contains at least one uncharged state. After an uncharged state in T_L is charged, T_{LP} still contains an uncharged state from T_P. The inductive step is similar. The result follows. □

Theorem 6 *Given any subset T of the NNFA states, we can compute all of the sets $\delta(T,a)$ for every alphabet symbols $a \in \Sigma$ in time $O(|T| + |\delta(T,\Sigma)|)$.*

Proof The sets belonging to $finddomain(T)$ are represented by all the nodes P_T along the paths from the states belonging to T to the roots of the *F-forest*. These nodes P_T can be found in $O(|T| + |P_T|)$ time by a marked traversal of parent pointers in the forest. Observe that $|P_T|$ can be much larger than $|T|$.

Computing $next\_states(T)$ involves two steps. First, for each node $n \in P_T$, we traverse a nonempty list of nodes in the *I-forest* representing $\{Y : [Fset(n), Y] \in lazy\delta\}$. This step takes time linear in the sum of the lengths of these lists. (Observe that this number can be much larger than $|P_T|$.) Second, if D_T is the set of all nodes in the *I-forest* belonging to these lists, then $next\_states(T) = \{Iset(n) : n \in D_T\}$. We can compute the set $next\_states(T)$ in $O(|\{[Fset(n), Y] : n \in P_T, [Fset(n), Y] \in lazy\delta\}|)$ time, which is $O(|T| + |\delta(T,\Sigma)|)$ time by Lemma 5.

Calculating $\delta(T,\Sigma)$ involves computing the union of the sets belonging to $next\_states(T)$. This is achieved in $O(|\delta(T,\Sigma)|)$ time using the left and right descendant pointers stored in each node belonging to D_T, traversing the unmarked leaves in the frontier, and marking leaves as they are traversed. Multiset discrimination

[7] can be used to separate out all of the sets $\{q \in \delta(T,\Sigma) | A(q) = a\}$ for each $a \in \Sigma$ in time $O(|\delta(T,\Sigma)|)$. □

Consider an NFA constructed from the following regular expression:

$$(\overbrace{\lambda|(\lambda|(\cdots(\lambda|a)^*)^*)\cdots)^*}^{k\ *'s})^n$$

In order to follow transitions labeled 'a', we have to examine $\Theta(n^2)$ edges and $\Theta(n)$ states in $\Theta(n^2)$ time for McNaughton and Yamada's NFA, $\Theta(kn)$ states and edges in $\Theta(kn)$ time for Thompson's machine, and $\Theta(n)$ states and edges in $\Theta(n)$ time for our compressed NFA.

7 Further Optimization

In this section, we introduce a simple transformation that can greatly improve the compressed NFA representation. If *lazy*δ contains both $[R,U]$ and $[S,U]$, and if there exists an *F-set* $T = R \cup S$, then we can replace $[R,U]$ and $[S,U]$ within *lazy*δ by a single pair $[T,U]$. Similarly, If *lazy*δ contains both $[U,R]$ and $[U,S]$, and if there exists an *I-set* $T = R \cup S$, then we can replace $[U,R]$ and $[U,S]$ within *lazy*δ by a single pair $[U,T]$. We call this technique *packing*. In a single linear time bottom up traversal of the *I-forest* and the *F-forest*, we can simplify *lazy*δ by packing. In the case of regular expression $(a_1|a_2|\cdots a_n)^*$ packing can simplify *lazy*δ from $3n-2$ pairs into a single pair. (see Figure 1.)

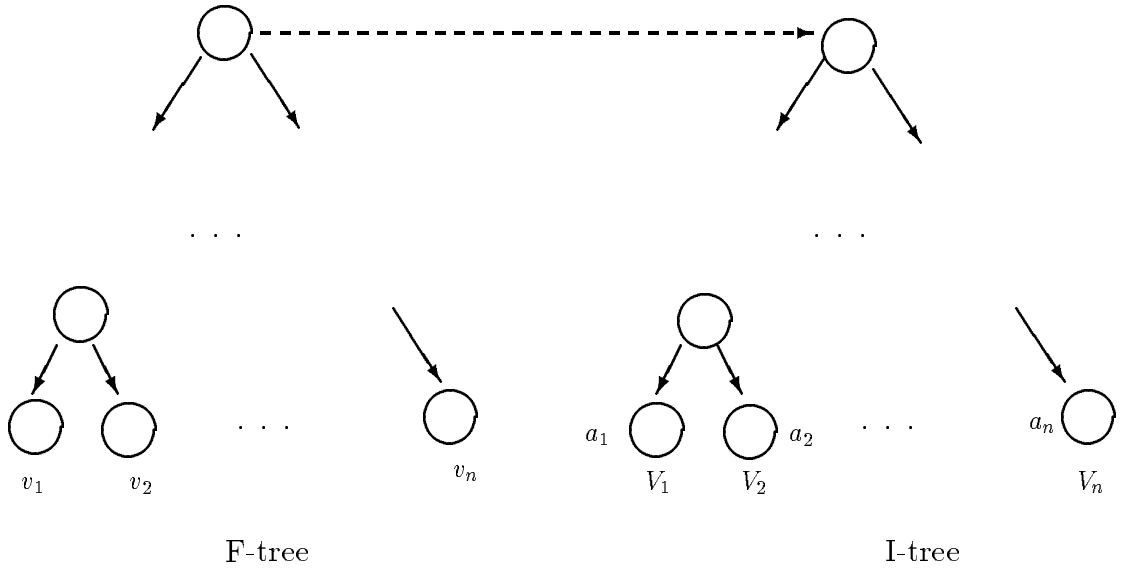

Figure 1: Compressed NFA of $(a_1|a_2|\cdots a_n)^*$ after packing F-sets and I-sets.

At the same time, we can carry out the same kind of path compression described in Section 6, so that the *F-forest* and *I-forest* only contain nodes in the domain (respectively range) of *lazy*δ. However, whereas

previously the forest leaves (corresponding to NFA states) were unaffected by compression, the packing transformation can remove leaves in the *F-forest* and *I-forest* from the domain and (respectively) range of $lazy\delta$. When path compression eliminates leaves, we need to turn the symbol assignment map A into a multi-valued mapping; that is, whenever leaves q1,...,qk are replaced by leaf q, we take the following steps;

- remove the old leaves q1,...,qk from the domain of A;
- assign the set of symbols $\{x : s \in \{q1, ..., qk\}, [s, x] \in A\}$ to A at q.

As an example of this, consider regular expression $(a_1|a_2|\cdots a_n)^*$ once again. Path compression will turn the data structure shown in Figure 1 into the one depicted in Figure 2. In using our compressed

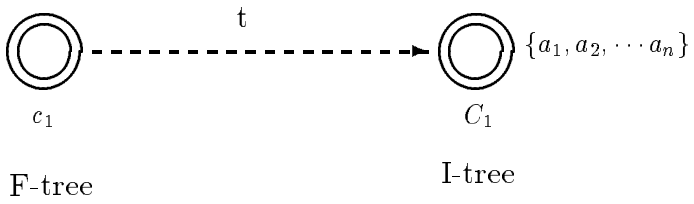

Figure 2: Compressed NFA of $(a_1|a_2|\cdots a_n)^*$ after Packing and Path Compression.

representation to simulate an NFA, the transition edge t (see Figure 2) can be taken only if the current transition symbol belongs to $\{a_1, a_2, \cdots a_n\}$ which labels node C_1.

Packing and path compression can not only speedup acceptance testing but improve DFA construction dramatically. In the remainder of this paper, we call our optimized compressed NFA representation CNFA.

8 Performance Benchmark

Experiments to benchmark the performance of the CNFA have been carried out for a range of regular expression patterns against a number of machines including Thompson's NFA, an optimized form of Thompson's NFA, and McNaughton and Yamada's NFA[13]. We build Thompson's NFA according to the construction rules described in [2]. Thompson's NFA usually contains excessively redundant states and λ-edges. However, to our knowledge there is no obvious/efficient algorithm to optimize Thompson's NFA without blowing up the linear space constraint. We therefore devise some simple but effective transformations that eliminate redundant states and edges in most of the test cases.

Our acceptance testing experiments show that the CNFA outperforms Thompson's NFA, Thompson's NFA optimized, and McNaughton and Yamada's NFA. For regular expression $(a|b\cdots)^*$, the CNFA is 12 times faster than Thompson's NFA, 2 times faster than Thompson's NFA optimized, and 50% faster than McNaughton and Yamada's NFA. For regular expression $((a|\lambda)(b|\lambda)\cdots)^*$, which is equivalent to $(a|b|\cdots)^*$,

the CNFA is 16 times faster than Thompson's NFA, 8 times faster than Thompson's NFA optimized, and 50% faster than McNaughton and Yamada's NFA. For regular expression $((a|\lambda)(b|\lambda)\cdots-)^*$, which accepts zero or more instances of an ordered string followed by a '$-$', the CNFA is 2 times faster than Thompson's NFA, 25% faster than Thompson's NFA optimized, but 80% slower than McNaughton and Yamada's NFA. For $((a|\lambda)^n-)^*$, the CNFA is comparable to Thompson's machine, 50% slower than Thompson's NFA optimized, and linearly faster than McNaughton and Yamada's NFA [1]. For $(abc\cdots)$ and $(abc\cdots)^*$, the CNFA is 75% slower than Thompson's NFA and McNaughton and Yamada's NFA, and 55% slower than Thompson's NFA optimized. However, acceptance testing for concatenation is quite fast for each of the NFA's being compared, and would not degrade our speedup ratio in general. Acceptance testing with a realistic programming language pattern shows that the CNFA is 7 times faster than Thompson's NFA, 60% faster than Thompson's NFA optimized, and 2 times faster than McNaughton and Yamada's NFA.

The benchmark for subset construction is more favorable. The CNFA outperforms the other machines not only in DFA construction time but also in constructed machine size. Subset construction is compared among the following five starting machines: the CNFA, Thompson's NFA, Thompson's NFA optimized, Thompson's NFA using important-state heuristic[2], and McNaughton and Yamada's NFA. Below is a high level modified specification of the classical Rabin and Scott subset construction for producing a DFA σ from an NFA δ:

$\sigma := \emptyset$
workset := $\{\{q_0\}\}$
while $\exists S \in$ workset do
    workset := workset - $\{S\}$
    for each symbol $a \in \Sigma$ and set of states $B = \{x \in \delta(S, \Sigma) | A(x) = a\}$ where $B \neq \emptyset$ do
        $\sigma(S, a) := B$
        $B :=$ ϵ-closure(B)
        if B does not belong to σ then
            workset := workset $\cup\{B\}$
        end if
    end for
end while

We implemented the preceding specification tailored to the CNFA and other machines. The only differences in these implementations is in the calculation of $\delta(S, \Sigma)$, where we use the efficient procedure described by Theorem 6, and in the ϵ-closure step, which is performed only by Thompson's NFA. The CNFA achieves

[1] McNaughton and Yamada's NFA suffers from a quadratic number of edges in this test pattern.

linear speedup and constructs a linearly smaller DFA in many of the test cases. See Figure 3 and 4 for a benchmark summary. The raw timing data is given in the Appendix. All the tests described in this paper are performed on a lightly loaded SUN 3/250 server. We used `getitimer()` and `setitimer()` primitives [19] to measure program execution time. It is interesting to note that the CNFA has a better speedup ratio on SUN Sparc based computers.

pattern	TNFA	TNFA (imp. state)	opt. TNFA	MYNFA
$(abc\cdots)^*$	5 times faster	comparable	comparable	comparable
$(a\|b\|\cdots)^*$	quadratic speedup	linear speedup	linear speedup	linear speedup
$(0\|1\cdots\|9)^n$	70 times faster	10 times faster	20% faster	10 times faster
$((a\|\lambda)(b\|\lambda)\cdots-)^*$	linear speedup	20% faster	linear speedup	5% faster
$((a\|\lambda)(b\|\lambda)\cdots)^*$	quadratic speedup	linear speedup	quadratic speedup	linear speedup
$(a\|b)^*a(a\|b)^n$	2.5 times faster	comparable	10% slower	50% faster
programming language	800 times faster	6 times faster	60% faster	6 times faster

Figure 3: CNFA Subset Construction Speedup Ratio

pattern	TNFA	TNFA (imp. state)	opt. TNFA	MYNFA
$(abc\cdots)^*$	comparable	comparable	comparable	comparable
$(a\|b\|c\cdots)^*$	linearly smaller	linearly smaller	comparable	linearly smaller
$(0\|1\|\cdots 9)^n$	200 times smaller	10 times smaller	comparable	10 times smaller
$((a\|\lambda)(b\|\lambda)\cdots-)^*$	3 times smaller	comparable	comparable	comparable
$((a\|\lambda)(b\|\lambda)\cdots)^*$	linearly smaller	linearly smaller	linearly smaller	linearly smaller
$(a\|b)^*a(a\|b)^n$	4 times smaller	comparable	comparable	comparable
programming language	10 times smaller	5 times smaller	20% larger	5 times smaller

Figure 4: DFA Size Improvement Ratio Starting from the CNFA

9 Conclusion

Theoretical analysis and confirming empirical evidence demonstrates that our proposed CNFA leads to a substantially more efficient way of turning regular expressions into DFA's (and minimum state DFA's in particular) than other NFA's in current use. It would be interesting future research to analyze the effect of packing and path compression on the CNFA. It would also be worthwhile to obtain a sharper analysis of the constant factors in comparing the CNFA with other NFA's.

References

[1] Aho, A., Hopcroft, J. and Ullman J., "Design and Analysis of Computer Algorithms", *Reading*, Addison-Wesley, 1974.

[2] Aho, A., Sethi, R. and Ullman, J., "Compilers Principles, Techniques, and Tools", *Reading*, Addison-Wesley, 1986.

[3] Aho, A., "Pattern Matching in Strings", in *Formal Language Theory*, ed. R. V. Book, Academic Press, Inc. 1980.

[4] Berry, G. and Cosserat, L., "The Esterel synchronous programming language and its mathematical semantics" in *Seminar in Concurrency*, S. D. Brookes, A. W. Roscoe, and G. Winskel, eds., LNCS 197, Springer-Verlag, 1985.

[5] Berry, G. and Sethi, R., "From Regular Expressions to Deterministic Automata" *Theoretical Computer Science*, 48 (1986), pp. 117-126.

[6] Brzozowski, J., "Derivatives of Regular Expressions", *JACM*, Vol. 11, No. 4., Oct. 1964, pp. 481-494.

[7] Cai, J. and Paige, R., "Look Ma, No Hashing, And No Arrays Neither", *ACM POPL*, Jan. 1991, pp. 143 - 154.

[8] Emerson, E. and Lei, C., "Model Checking in the Propositional Mu-Calculus", *Proc. IEEE Conf. on Logic in Computer Science*, 1986, pp. 86 - 106.

[9] Fredman, M., Komlos, J., and Szemeredi, E., "Storing a Sparse Table with O(1) Worst Case Access Time", *JACM*, vol. 31, no. 3, pp. 538-544, July, 1984.

[10] Hopcroft, J. and Ullman, J., "Formal Languages and Their Relation to Automata",*Reading*, Addison-Wesley, 1969.

[11] Kleene, S., "Representation of events in nerve nets and finite automata", in *Automata Studies, Ann. Math. Studies No. 34*, Princeton U. Press, 1956, pp. 3 - 41.

[12] Knuth, D., "On the translation of languages from left to right", *Information and Control*, Vol. 8, Num. 6, 1965, pp. 607 - 639.

[13] McNaughton, R. and Yamada, H. "Regular Expressions and State Graphs for Automata", *IRA Trans. on Electronic Computers*, Vol. EC-9, No. 1, Mar. 1960, pp 39-47.

[14] Myhill, J., "Finite automata and representation of events," WADC, Tech. Rep. 57-624, 1957.

[15] Nerode, A., "Linear automaton transformations," *Proc. Amer. Math Soc.*, Vol. 9, pp. 541 - 544, 1958.

[16] Rabin, M. and Scott, D., "Finite automata and their decision problems" *IBM J. Res. Develop.*, Vol. 3, No. 2, Apr., 1959, pp. 114 - 125.

[17] Ritchie, D. and Thompson, K. "The UNIX Time-Sharing System" *Communication ACM*, Vol. 17, No. 7, Jul., 1974, pp. 365 - 375.

[18] Thompson, K., "Regular Expression search Algorithm", *Communication ACM* 11:6 (1968), pp. 419-422.

[19] "SunOS Reference Manual VOL. II", *Programmer's Manual*, SUN microsystems, 1989.

[20] Ullman, J., "Computational Aspects of VLSI", *Computer Science Press*, 1984.

APPENDIX: Benchmark Results [2]

Acceptance Testing Benchmark

$(abc\cdots)$

length	TNFA	TNFA	unopt. CNFA	CNFA	MYNFA	TNFA vs CNFA	opt. TNFA vs CNFA	MYNFA vs CNFA	unopt. CNFA vs CNFA
1000	0.14	0.34	0.58	0.76	0.18	0.18	0.45	0.24	0.76
1500	0.20	0.52	1.00	1.18	0.30	0.18	0.44	0.25	0.84
2000	0.30	0.74	1.32	1.58	0.42	0.19	0.47	0.27	0.84
2500	0.38	0.90	1.60	2.00	0.54	0.19	0.45	0.27	0.80
3000	0.44	1.12	2.00	2.44	0.66	0.18	0.46	0.27	0.82
4000	0.64	1.54	2.58	3.32	0.84	0.19	0.46	0.25	0.78
4500	0.72	1.56	2.78	3.42	0.96	0.21	0.46	0.28	0.82
5000	0.70	1.48	2.88	3.56	0.92	0.20	0.42	0.26	0.81

$(abc\cdots)^*$

length	TNFA	TNFA	unopt. CNFA	CNFA	MYNFA	TNFA vs CNFA	opt. TNFA vs CNFA	MYNFA vs CNFA	unopt. CNFA vs CNFA
10	0.32	0.62	1.10	1.56	0.32	0.21	0.40	0.21	0.71
20	0.26	0.60	1.12	1.44	0.34	0.18	0.42	0.24	0.78
30	0.28	0.64	1.12	1.42	0.36	0.20	0.45	0.25	0.79
40	0.28	0.64	1.08	1.44	0.34	0.19	0.44	0.24	0.75
50	0.26	0.64	1.08	1.48	0.38	0.18	0.43	0.26	0.73
60	0.28	0.64	1.12	1.50	0.34	0.19	0.43	0.23	0.75
70	0.26	0.66	1.10	1.46	0.34	0.18	0.45	0.23	0.75
80	0.26	0.64	1.10	1.46	0.32	0.18	0.44	0.22	0.75
90	0.28	0.64	1.14	1.46	0.36	0.19	0.44	0.25	0.78
100	0.26	0.64	1.10	1.46	0.34	0.18	0.44	0.23	0.75

$(a|b|c\cdots)^*$

length	TNFA	TNFA	unopt. CNFA	CNFA	MYNFA	TNFA vs CNFA	opt. TNFA vs CNFA	MYNFA vs CNFA	unopt. CNFA vs CNFA
10	5.46	1.36	4.20	1.76	0.82	3.10	0.77	0.47	2.39
20	10.48	2.18	7.52	2.02	1.38	5.19	1.08	0.68	3.72
30	15.70	3.04	10.86	2.18	1.86	4.85	1.39	0.85	4.98
40	21.16	3.76	14.28	2.56	2.42	8.27	1.47	0.95	5.58
50	26.22	4.60	17.28	2.84	3.00	9.23	1.62	1.06	6.08
60	31.62	5.46	22.56	3.12	3.66	10.13	1.75	1.17	7.23
70	36.62	6.20	23.94	3.26	4.36	11.23	1.90	1.34	7.34
80	42.02	7.12	27.38	3.56	5.22	11.24	2.00	1.47	7.69
90	47.94	7.92	30.44	3.90	6.00	12.29	2.03	1.54	7.81
100	52.00	8.70	35.10	4.10	6.88	12.68	2.12	1.68	8.56

[2] All tests are performed on a SUN 3/250 server. Benchmark time is in seconds.

$$((a|\lambda)^n -)^*$$

length	TNFA	TNFA	unopt. CNFA	CNFA	MYNFA	TNFA vs CNFA	opt. TNFA vs CNFA	MYNFA vs CNFA	unopt. CNFA vs CNFA
10	7.14	4.30	5.50	8.10	3.96	0.88	0.53	0.49	0.70
20	12.94	7.14	9.14	13.76	12.14	0.94	0.52	0.88	0.69
30	19.90	10.60	13.76	20.74	26.12	0.96	0.51	1.26	0.66
40	25.92	12.90	17.06	26.22	42.16	0.99	0.49	1.61	0.65
50	31.46	16.82	22.36	34.26	66.54	0.92	0.49	1.94	0.65
60	36.10	18.96	24.98	39.78	91.74	0.91	0.48	2.31	0.63
70	43.56	22.96	29.54	46.04	127.28	0.95	0.50	2.76	0.64
80	51.18	25.96	35.20	53.60	171.02	0.95	0.48	3.19	0.66
90	52.66	26.80	35.54	54.24	187.56	1.01	0.51	3.46	0.68
100	61.30	31.12	41.00	63.44	248.04	0.97	0.49	3.91	0.65

$$((a|\lambda)(b|\lambda)\cdots)^*$$

length	TNFA	TNFA	unopt. CNFA	CNFA	MYNFA	TNFA vs CNFA	opt. TNFA vs CNFA	MYNFA vs CNFA	unopt. CNFA vs CNFA
10	7.08	3.92	4.26	1.94	0.86	3.56	2.02	0.44	2.20
20	13.06	7.42	7.60	2.06	1.38	6.34	3.60	0.67	3.69
30	19.92	10.96	10.78	2.30	1.92	8.66	4.77	0.83	4.69
40	26.32	14.38	14.16	2.52	2.44	10.44	5.71	0.97	5.62
50	32.32	18.00	17.34	2.84	3.00	11.38	6.34	1.06	6.11
60	37.68	21.66	20.78	3.10	3.66	12.15	6.99	1.18	6.70
70	43.82	25.12	24.06	3.24	4.34	13.52	7.75	1.34	7.42
80	51.54	28.48	27.78	3.58	5.18	14.40	7.96	1.45	7.75
90	57.80	32.08	30.80	3.88	5.90	14.89	8.27	1.52	7.94
100	64.56	35.46	33.98	4.06	6.86	15.90	8.73	1.69	8.37

$$((a|\lambda)(b|\lambda)\cdots -)^*$$

length	TNFA	TNFA	unopt. CNFA	CNFA	MYNFA	TNFA vs CNFA	opt. TNFA vs CNFA	MYNFA vs CNFA	unopt. CNFA vs CNFA
10	4.40	2.82	2.66	3.36	0.68	1.31	0.84	0.20	0.79
20	8.08	4.94	4.08	4.86	1.00	1.72	1.02	0.21	0.83
30	12.30	7.16	5.50	6.54	1.34	1.88	1.09	0.20	0.84
40	16.06	9.22	6.72	7.96	1.64	2.01	1.16	0.21	0.87
50	19.22	11.24	8.10	9.34	1.88	2.05	1.20	0.20	0.87
60	23.46	13.90	9.80	11.04	2.38	2.13	1.26	0.22	0.89
70	27.32	16.18	11.22	12.68	2.84	2.15	1.28	0.22	0.89
80	31.90	18.18	12.72	14.34	3.16	2.22	1.27	0.22	0.89
90	34.80	19.92	13.64	15.34	3.40	2.27	1.30	0.22	0.89
100	38.98	22.04	14.96	18.50	3.78	2.10	1.19	0.20	0.81

Subset Construction Benchmark

$$(abc\cdots)^*$$

Construction Time

length	TNFA	TNFA (imp. state)	opt. TNFA	unopt. CNFA	CNFA	MYNFA	TNFA (imp. state) vs CNFA	opt. TNFA vs CNFA	MYNFA vs CNFA
100	0.04	0.02	0.02	0.04	0.02	0.02	1.00	1.00	1.00
200	0.12	0.04	0.06	0.06	0.04	0.06	1.00	1.50	1.50
300	0.16	0.06	0.04	0.06	0.04	0.08	1.50	1.00	2.00
400	0.28	0.10	0.08	0.08	0.10	0.08	1.00	0.80	0.80
500	0.40	0.12	0.10	0.12	0.12	0.10	1.00	0.83	0.83
600	0.56	0.12	0.14	0.14	0.14	0.14	0.86	1.00	1.00
700	0.74	0.14	0.18	0.14	0.14	0.20	1.00	1.29	1.43
800	1.04	0.20	0.20	0.20	0.18	0.24	1.11	1.11	1.33
900	1.12	0.18	0.22	0.22	0.18	0.22	1.00	1.22	1.22
1000	1.72	0.26	0.20	0.20	0.22	0.22	1.18	0.91	1.00

Constructed DFA Size

machine	length	node no	edge no	node weight	edge weight	length	node no	edge no	node weight	edge weight
TNFA	100	101	101	105	103	200	201	201	205	203
TNFA (imp. state)	100	101	101	101	101	200	201	201	201	201
opt. TNFA	100	100	100	100	100	200	200	200	200	200
unopt. CNFA	100	101	101	101	101	200	201	201	201	201
CNFA	100	101	101	101	101	200	201	201	201	201
MYNFA	100	101	101	101	101	200	201	201	201	201
TNFA	300	301	301	305	303	400	401	401	405	403
TNFA (imp. state)	300	301	301	301	301	400	401	401	401	401
opt. TNFA	300	300	300	300	300	400	400	400	400	400
unopt. CNFA	300	301	301	301	301	400	401	401	401	401
CNFA	300	301	301	301	301	400	401	401	401	401
MYNFA	300	301	301	301	301	400	401	401	401	401
TNFA	500	501	501	505	503	600	601	601	605	603
TNFA (imp. state)	500	501	501	501	501	600	601	601	601	601
opt. TNFA	500	500	500	500	500	600	600	600	600	600
unopt. CNFA	500	501	501	501	501	600	601	601	601	601
CNFA	500	501	501	501	501	600	601	601	601	601
MYNFA	500	501	501	501	501	600	601	601	601	601
TNFA	700	701	701	705	703	800	801	801	805	803
TNFA (imp. state)	700	701	701	701	701	800	801	801	801	801
opt. TNFA	700	700	700	700	700	800	800	800	800	800
unopt. CNFA	700	701	701	701	701	800	801	801	801	801
CNFA	700	701	701	701	701	800	801	801	801	801
MYNFA	700	701	701	701	701	800	801	801	801	801
TNFA	900	901	901	905	903	1000	1001	1001	1005	1003
TNFA (imp. state)	900	901	901	901	901	1000	1001	1001	1005	1003
opt. TNFA	900	900	900	900	900	1000	1000	1000	1000	1000
unopt. CNFA	900	901	901	901	901	1000	1001	1001	1005	1003
CNFA	900	901	901	901	901	1000	1001	1001	1005	1003
MYNFA	900	901	901	901	901	1000	1001	1001	1005	1003

$$(a|b|c\cdots)^*$$

Construction Time

length	TNFA	TNFA (imp. state)	opt. TNFA	unopt. CNFA	CNFA	MYNFA	TNFA (imp. state) vs CNFA	opt. TNFA vs CNFA	MYNFA vs CNFA
10	0.08	0.02	0.00	0.02	0.00	0.02	-	1.00	-
20	0.92	0.06	0.00	0.06	0.02	0.04	3.00	0.00	2.00
30	4.44	0.08	0.02	0.08	0.02	0.08	4.00	1.00	4.00
40	12.70	0.14	0.00	0.18	0.00	0.16	-	1.00	-
50	29.94	0.24	0.02	0.24	0.02	0.28	12.00	1.00	14.00
60	61.18	0.46	0.02	0.36	0.02	0.30	23.00	1.00	15.00
70	111.88	0.50	0.04	0.46	0.02	0.46	25.00	2.00	23.00
80	188.74	0.64	0.04	0.54	0.02	0.58	32.00	2.00	29.00
90	300.72	0.78	0.06	0.78	0.02	0.74	39.00	3.00	37.00
100	450.90	0.96	0.06	0.94	0.02	0.92	48.00	3.00	46.00

Constructed DFA Size

machine	length	node no	edge no	node weight	edge weight	length	node no	edge no	node weight	edge weight
TNFA	10	11	110	285	2904	20	21	420	1070	21609
TNFA (imp. state)	10	11	110	11	110	20	21	420	21	420
opt. TNFA	10	1	10	1	10	20	1	20	1	20
unopt. CNFA	10	11	110	11	110	20	21	420	21	420
CNFA	10	2	20	2	20	20	2	40	2	40
MYNFA	10	11	110	11	110	20	21	420	21	420
TNFA	30	31	930	2355	71114	40	41	1640	4140	166419
TNFA (imp. state)	30	31	930	31	930	40	41	1640	41	1640
opt. TNFA	30	1	30	1	30	40	1	40	1	40
unopt. CNFA	30	31	930	31	930	40	41	1640	41	1640
CNFA	30	2	60	2	60	40	2	80	2	80
MYNFA	30	31	930	31	930	40	41	1640	41	1640
TNFA	50	51	2550	6425	322524	60	61	3660	9210	554429
TNFA (imp. state)	50	51	2550	51	2550	60	61	3660	61	3660
opt. TNFA	50	1	50	1	50	60	1	60	1	60
unopt. CNFA	50	51	2550	51	2550	60	61	3660	61	3660
CNFA	50	2	100	2	100	60	2	120	2	120
MYNFA	50	51	2550	51	2550	60	61	3660	61	3660
TNFA	70	71	4970	12495	877134	80	81	6480	16280	1305639
TNFA (imp. state)	70	71	4970	71	4970	80	81	6480	81	6480
opt. TNFA	70	1	70	1	70	80	1	80	1	80
unopt. CNFA	70	71	4970	71	4970	80	81	6480	81	6480
CNFA	70	2	140	2	140	80	2	160	2	160
MYNFA	70	71	4970	71	4970	80	81	6480	81	6480
TNFA	90	91	8190	20565	1854944	100	101	10100	25350	2540049
TNFA (imp. state)	90	91	8190	91	8190	100	101	10100	101	10100
opt. TNFA	90	1	90	1	90	100	1	100	1	100
unopt. CNFA	90	91	8190	91	8190	100	101	10100	101	10100
CNFA	90	2	180	2	180	100	2	200	2	200
MYNFA	90	91	8190	91	8190	100	101	10100	101	10100

$$(0|1|\cdots 9)^n$$

Construction Time

length	TNFA	TNFA (imp. state)	opt. TNFA	unopt. CNFA	CNFA	MYNFA	TNFA (imp. state) vs CNFA	opt. TNFA vs CNFA	MYNFA vs CNFA
25	1.54	0.28	0.04	0.20	0.02	0.26	14.00	2.00	13.00
50	3.16	0.48	0.06	0.44	0.04	0.52	12.00	1.50	13.00
75	4.82	0.78	0.08	0.68	0.06	0.80	13.00	1.33	13.33
100	6.50	1.02	0.12	0.90	0.12	1.00	8.50	1.00	8.33
125	8.16	1.26	0.14	1.08	0.10	1.20	12.60	1.40	12.00
150	9.80	1.60	0.18	1.38	0.12	1.54	13.33	1.50	12.83
175	11.42	1.70	0.20	1.46	0.14	1.72	12.14	1.43	12.29
200	13.20	2.06	0.24	1.74	0.18	1.94	11.44	1.33	10.78
225	14.98	2.34	0.24	2.28	0.20	2.32	11.70	1.20	11.60
250	16.30	2.58	0.28	2.52	0.22	2.70	11.73	1.27	12.27

Constructed DFA Size

machine	length	node no	edge no	node weight	edge weight	length	node no	edge no	node weight	edge weight
TNFA	25	251	2410	5939	57004	50	501	4910	12039	118004
TNFA (imp. state)	25	251	2410	251	2410	50	501	4910	501	4910
opt. TNFA	25	26	250	26	250	50	51	500	51	500
unopt. CNFA	25	251	2410	251	2410	50	501	4910	501	4910
CNFA	25	26	250	26	250	50	51	500	51	500
MYNFA	25	251	2410	251	2410	50	501	4910	501	4910
TNFA	75	751	7410	18139	179004	100	1001	9910	24239	240004
TNFA (imp. state)	75	751	7410	751	7410	100	1001	9910	1001	9910
opt. TNFA	75	76	750	76	750	100	101	1000	101	1000
unopt. CNFA	75	751	7410	751	7410	100	1001	9910	1001	9910
CNFA	75	76	750	76	750	100	101	1000	101	1000
MYNFA	75	751	7410	751	7410	100	1001	9910	1001	9910
TNFA	125	1251	12410	30339	301004	150	1501	14910	36439	362004
TNFA (imp. state)	125	1251	12410	1251	12410	150	1501	14910	1501	14910
opt. TNFA	125	126	1250	126	1250	150	151	1500	151	1500
unopt. CNFA	125	1251	12410	1251	12410	150	1501	14910	1501	14910
CNFA	125	126	1250	126	1250	150	151	1500	151	1500
MYNFA	125	1251	12410	1251	12410	150	1501	14910	1501	14910
TNFA	175	1751	17410	42539	423004	200	2001	19910	48639	484004
TNFA (imp. state)	175	1751	17410	1751	17410	200	2001	19910	2001	19910
opt. TNFA	175	176	1750	176	1750	200	201	2000	201	2000
unopt. CNFA	175	1751	17410	1751	17410	200	2001	19910	2001	19910
CNFA	175	176	1750	176	1750	200	201	2000	201	2000
MYNFA	175	1751	17410	1751	17410	200	2001	19910	2001	19910
TNFA	225	2251	22410	54739	545004	250	2501	24910	60839	606004
TNFA (imp. state)	225	2251	22410	2251	22410	250	2501	24910	2501	24910
opt. TNFA	225	226	2250	226	2250	250	251	2500	251	2500
unopt. CNFA	225	2251	22410	2251	22410	250	2501	24910	2501	24910
CNFA	225	226	2250	226	2250	250	251	2500	251	2500
MYNFA	225	2251	22410	2251	22410	250	2501	24910	2501	24910

$$((a|\lambda)(b|\lambda)\cdots-)^*$$

Construction Time

length	TNFA	TNFA (imp. state)	opt. TNFA	unopt. CNFA	CNFA	MYNFA	TNFA (imp. state) vs CNFA	opt. TNFA vs CNFA	MYNFA vs CNFA
25	0.34	0.04	0.10	0.04	0.04	0.04	1.00	2.50	1.00
50	3.86	0.14	0.40	0.12	0.12	0.14	1.17	3.33	1.17
75	18.16	0.30	1.34	0.24	0.28	0.28	1.07	4.79	1.00
100	55.50	0.48	2.90	0.44	0.46	0.50	1.04	6.30	1.09
125	132.52	0.80	5.72	0.66	0.76	0.74	1.05	7.53	0.97
150	270.34	1.14	9.24	0.96	1.02	1.06	1.12	9.06	1.04
175	496.94	1.54	14.64	1.26	1.38	1.50	1.12	10.61	1.09
200	839.94	2.04	20.94	1.62	1.76	1.88	1.16	11.90	1.07
225	1392.42	3.20	32.02	2.12	2.40	2.44	1.33	13.34	1.02
250	2065.14	3.16	41.62	2.78	2.74	2.90	1.15	15.19	1.06

Constructed DFA Size

machine	length	node no	edge no	node weight	edge weight	length	node no	edge no	node weight	edge weight
TNFA	25	27	351	1027	8476	50	52	1281	3928	65076
TNFA (imp. state)	25	27	351	27	351	50	52	1281	53	1326
opt. TNFA	25	27	351	27	351	50	52	1281	53	1326
unopt. CNFA	25	27	351	27	351	50	52	1281	53	1326
CNFA	25	27	351	27	351	50	52	1281	53	1326
MYNFA	25	27	351	27	351	50	52	1281	53	1326
TNFA	75	77	2881	8703	216676	100	102	5106	15353	510151
TNFA (imp. state)	75	77	2881	78	2926	100	102	5106	103	5151
opt. TNFA	75	77	2881	78	2926	100	102	5106	103	5151
unopt. CNFA	75	77	2881	78	2926	100	102	5106	103	5151
CNFA	75	77	2881	78	2926	100	102	5106	103	5151
MYNFA	75	77	2881	78	2926	100	102	5106	103	5151
TNFA	125	127	7956	23878	992376	150	152	11431	34278	1710226
TNFA (imp. state)	125	127	7956	128	8001	150	152	11431	153	11476
opt. TNFA	125	127	7956	128	8001	150	152	11431	153	11476
unopt. CNFA	125	127	7956	128	8001	150	152	11431	153	11476
CNFA	125	127	7956	128	8001	150	152	11431	153	11476
MYNFA	125	127	7956	128	8001	150	152	11431	153	11476
TNFA	175	177	15531	46553	2710576	200	202	20256	60703	4040301
TNFA (imp. state)	175	177	15531	178	15576	200	202	20256	203	20301
opt. TNFA	175	177	15531	178	15576	200	202	20256	203	20301
unopt. CNFA	175	177	15531	178	15576	200	202	20256	203	20301
CNFA	175	177	15531	178	15576	200	202	20256	203	20301
MYNFA	175	177	15531	178	15576	200	202	20256	203	20301
TNFA	225	227	25606	76728	5746276	250	252	31581	94628	7875376
TNFA (imp. state)	225	227	25606	228	25651	250	252	31581	253	31626
opt. TNFA	225	227	25606	228	25651	250	252	31581	253	31626
unopt. CNFA	225	227	25606	228	25651	250	252	31581	253	31626
CNFA	225	227	25606	228	25651	250	252	31581	253	31626
MYNFA	225	227	25606	228	25651	250	252	31581	253	31626

$$((a|\lambda)(b|\lambda)\cdots)^*$$

Construction Time

length	TNFA	TNFA (imp. state)	opt. TNFA	unopt. CNFA	CNFA	MYNFA	TNFA (imp. state) vs CNFA	opt. TNFA vs CNFA	MYNFA vs CNFA
10	0.14	0.02	0.04	0.02	0.00	0.02	-	-	-
20	1.44	0.04	0.14	0.04	0.00	0.06	-	-	-
30	6.24	0.10	0.28	0.10	0.02	0.06	5.00	14.00	3.00
40	19.20	0.16	0.58	0.14	0.00	0.18	-	-	-
50	45.64	0.26	1.10	0.22	0.02	0.24	13.00	55.00	12.00
60	93.00	0.36	1.84	0.32	0.02	0.36	18.00	92.00	18.00
70	170.80	0.48	2.88	0.42	0.02	0.48	24.00	144.00	24.00
80	287.48	0.68	4.24	0.56	0.02	0.62	34.00	212.00	31.00
90	457.08	0.78	5.88	0.72	0.02	0.74	39.00	294.00	37.00
100	693.54	0.90	8.00	0.92	0.02	1.06	45.00	400.00	53.00

Constructed DFA Size

machine	length	node no	edge no	node weight	edge weight	length	node no	edge no	node weight	edge weight
TNFA	10	11	110	363	3630	20	21	420	1323	26460
TNFA (imp. state)	10	11	110	11	110	20	21	420	21	420
opt. TNFA	10	10	100	10	100	20	20	400	20	400
unopt. CNFA	10	11	110	11	110	20	21	420	21	420
CNFA	10	2	20	2	20	20	2	40	2	40
MYNFA	10	11	110	11	110	20	21	420	21	420
TNFA	30	31	930	2883	86490	40	41	1640	5043	201720
TNFA (imp. state)	30	31	930	31	930	40	41	1640	41	1640
opt. TNFA	30	30	900	30	900	40	40	1600	40	1600
unopt. CNFA	30	31	930	31	930	40	41	1640	41	1640
CNFA	30	2	60	2	60	40	2	80	2	80
MYNFA	30	31	930	31	930	40	41	1640	41	1640
TNFA	50	51	2550	7803	390150	60	61	3660	11163	669780
TNFA (imp. state)	50	51	2550	51	2550	60	61	3660	61	3660
opt. TNFA	50	50	2500	50	2500	60	60	3600	60	3600
unopt. CNFA	50	51	2550	51	2550	60	61	3660	61	3660
CNFA	50	2	100	2	100	60	2	120	2	120
MYNFA	50	51	2550	51	2550	60	61	3660	61	3660
TNFA	70	71	4970	15123	1058610	80	81	6480	19683	1574640
TNFA (imp. state)	70	71	4970	71	4970	80	81	6480	81	6480
opt. TNFA	70	70	4900	70	4900	80	80	6400	80	6400
unopt. CNFA	70	71	4970	71	4970	80	81	6480	81	6480
CNFA	70	2	140	2	140	80	2	160	2	160
MYNFA	70	71	4970	71	4970	80	81	6480	81	6480
TNFA	90	91	8190	24843	2235870	100	101	10100	30603	3060300
TNFA (imp. state)	90	91	8190	91	8190	100	101	10100	101	10100
opt. TNFA	90	90	8100	90	8100	100	100	10000	100	10000
unopt. CNFA	90	91	8190	91	8190	100	101	10100	101	10100
CNFA	90	2	180	2	180	100	2	200	2	200
MYNFA	90	91	8190	91	8190	100	101	10100	101	10100

$$(a|b)^*a(a|b)^n$$

Construction Time

length	TNFA	TNFA (imp. state)	opt. TNFA	unopt. CNFA	CNFA	MYNFA	TNFA (imp. state) vs CNFA	opt. TNFA vs CNFA	MYNFA vs CNFA
1	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	1.00
2	0.02	0.02	0.02	0.00	0.02	0.00	1.00	1.00	0.00
3	0.04	0.02	0.02	0.00	0.02	0.00	1.00	1.00	0.00
4	0.02	0.02	0.02	0.00	0.02	0.02	1.00	1.00	1.00
5	0.06	0.02	0.04	0.02	0.02	0.04	1.00	2.00	2.00
6	0.16	0.04	0.06	0.06	0.06	0.08	0.67	1.00	1.33
7	0.30	0.14	0.12	0.14	0.14	0.14	1.00	0.86	1.00
8	0.70	0.26	0.22	0.30	0.28	0.36	0.93	0.79	1.29
9	1.58	0.54	0.50	0.76	0.56	0.94	0.96	0.89	1.68
10	3.54	1.14	1.06	2.26	1.22	3.34	0.93	0.87	2.74

Constructed DFA Size

machine	length	node no	edge no	node weight	edge weight	length	node no	edge no	node weight	edge weight
TNFA	1	5	10	39	83	2	9	18	89	183
TNFA (imp. state)	1	5	10	9	19	2	9	18	21	43
opt. TNFA	1	4	8	8	16	2	8	16	20	40
unopt. CNFA	1	5	10	9	19	2	9	18	21	43
CNFA	1	5	10	9	19	2	9	18	21	43
MYNFA	1	5	10	9	19	2	9	18	21	43
TNFA	3	17	34	205	415	4	33	66	469	943
TNFA (imp. state)	3	17	34	49	99	4	33	66	113	227
opt. TNFA	3	16	32	48	96	4	32	64	112	224
unopt. CNFA	3	17	34	49	99	4	33	66	113	227
CNFA	3	17	34	49	99	4	33	66	113	227
MYNFA	3	17	34	49	99	4	33	66	113	227
TNFA	5	65	130	1061	2127	6	129	258	2373	4751
TNFA (imp. state)	5	65	130	257	515	6	129	258	577	1155
opt. TNFA	5	64	128	256	512	6	128	256	576	1152
unopt. CNFA	5	65	130	257	515	6	129	258	577	1155
CNFA	5	65	130	257	515	6	129	258	577	1155
MYNFA	5	65	130	257	515	6	129	258	577	1155
TNFA	7	257	514	5253	10511	8	513	1026	11525	23055
TNFA (imp. state)	7	257	514	1281	2563	8	513	1026	2817	5635
opt. TNFA	7	256	512	1280	2560	8	512	1024	2816	5632
unopt. CNFA	7	257	514	1281	2563	8	513	1026	2817	5635
CNFA	7	257	514	1281	2563	8	513	1026	2817	5635
MYNFA	7	257	514	1281	2563	8	513	1026	2817	5635
TNFA	9	1025	2050	25093	50191	10	2049	4098	54277	108559
TNFA (imp. state)	9	1025	2050	6145	12291	10	2049	4098	13313	26627
opt. TNFA	9	1024	2048	6144	12288	10	2048	4096	13312	26624
unopt. CNFA	9	1025	2050	6145	12291	10	2049	4098	13313	26627
CNFA	9	1025	2050	6145	12291	10	2049	4098	13313	26627
MYNFA	9	1025	2050	6145	12291	10	2049	4098	13313	26627